CS 3120 Quiz Day 2

This packet contains the quizzes for this quiz day. This **cover sheet** is here to provide instructions, and to cover the questions until the quiz begins. **do not remove this cover sheet** until your proctor instructs you to do so.

You will have the entire class period to complete these quizzes. Each quiz is two pages (front and back of one sheet of paper) worth of questions. Make sure to **write your name and computing id at the top of each individual quiz**.

When you are done, you will come to the front of the room and cut off the staple to this quiz booklet. Afterward, you will discard this cover sheet and submit each quiz separately in a different pile. The proctors will be available at the front of the room to clarify this if you have any questions.

This quiz is CLOSED text book, closed-notes, closed-calculator, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

In theory, there is no difference between theory and practice.
But, in practice, there is.

THIS COVER SHEET WILL NOT BE SUBMITTED. DO NOT PUT WORK YOU WANT GRADED ON THIS PAGE

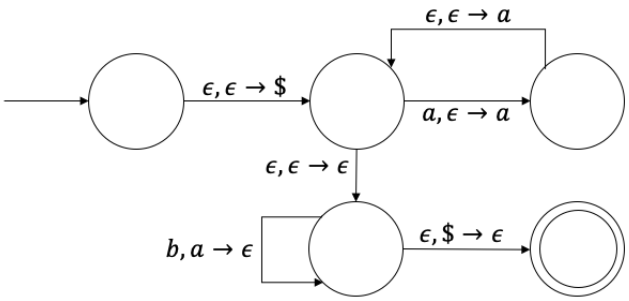
Quiz - Module 3: Context-Free Grammars

Name _____

1. [8 points] Answer the following True/False questions.

Context-free languages are recognized by *non-deterministic pushdown automata*	**True**	**False**
In a *CFG*, it is okay to have a symbol be both a *variable* and a *terminal* (e.g., $A \in V \wedge A \in \Sigma$)	**True**	**False**
Java is a *context-free grammar* because no single line of code's validity is dependent on a previous line of code	**True**	**False**
If a *pushdown automata* runs out of stack space, then the machine *rejects*	**True**	**False**
A *pushdown automata* MUST push xor pop from the stack on each transition	**True**	**False**
A *PDA* can have a different *stack alphabet* (Γ) than *input alphabet*(Σ)	**True**	**False**
Every *pushdown automata* can be converted into a *context-free grammar* that generates the same language	**True**	**False**
Using the *pumping lemma*, we can show that $D = \{ww | w \in \{0,1\}^*\}$ is not context-free	**True**	**False**

2. [3 points] Write an expression for the language recognized by the following *pushdown automata*. Please use the variable n if you need a variable.



3. [3 points] Consider the *CFG* for this language: *Bitstrings that contain 01 OR contain only 0s OR contain only 1s*. Complete the grammar below. Note that S is the start variable. *Hint: A and D handle the case where the string contains a 01 while B and C handle the other cases.*

$S \rightarrow A|B|C$

$A \rightarrow$ _____

$B \rightarrow$ _____$|0|\epsilon$

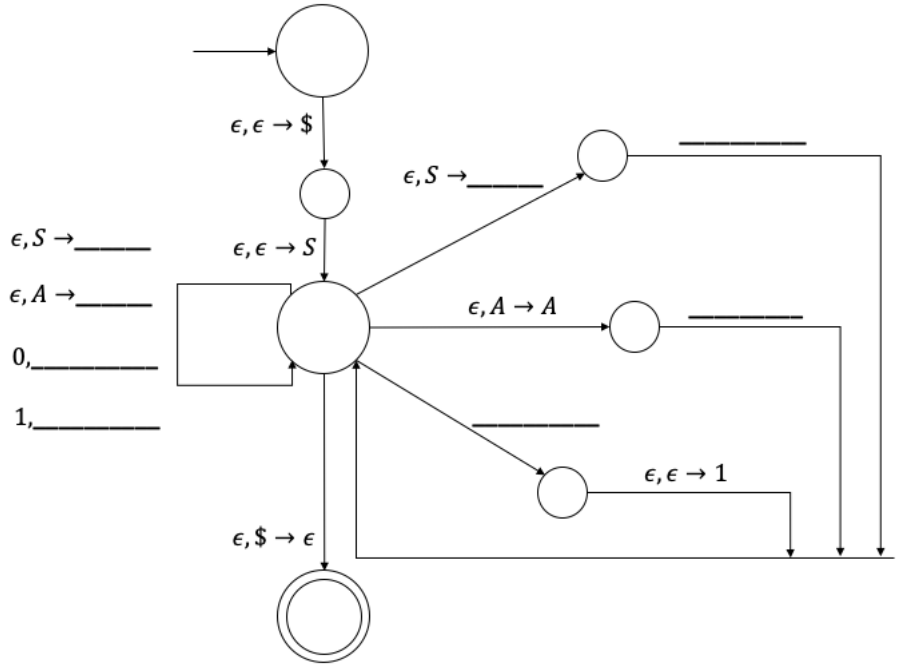
$C \rightarrow 1C|1|\epsilon$

$D \rightarrow 0D|$_____$|\epsilon$

In class, we proved the following claim: *If L is a context-free grammar, then there exists some pushdown automata that recognizes the same language as L.* For this question, you will show you understand this proof by manually converting a specific grammar into an equivalent finite automata.

4. [6 points] Take a look at the image below. On the left you are given a *context-free grammar*. On the right, you are given a pushdown automata with missing transition labels. Fill in the labels to complete the *PDA* so that the machine is equivalent to the provided grammar. **Note that there are eight blanks to fill in!**

$S \rightarrow A \mid AB$
$A \rightarrow 0A \mid \epsilon$
$B \rightarrow 11$



NOTHING BELOW THIS POINT WILL BE GRADED

Quiz - Module 4: Turing Machines

Name _____

1. [8 points] Answer the following True/False questions.

Turing machines can sometimes loop forever, while *DFAs* can not	**True**	**False**
Recognizing whether a Turing machine loops forever is *NOT Turing-recognizable*	**True**	**False**
The *complement of a language L* is always as easy a problem to solve (decide or recognize) as the language L	**True**	**False**
Turing machines read in all of their input before deciding to accept or reject	**True**	**False**
A *Turing machine* transitions to a new state based on both the contents of the tape (at the head position) and current state of the machine	**True**	**False**
Suppose I designed a *Turing machine* that could move the head any odd number of tape positions (1,3,5,...) in a single step. This machine would be more powerful than a traditional *Turing machine*	**True**	**False**
Turing machines can read over the input multiple times if they want to	**True**	**False**
If a language L is *Turing recognizable* and *Co-Turing recognizable*, then we can construct a decider for L	**True**	**False**

The next three questions are about the proof from class that all *non-deterministic Turing machines* can be simulated using an equivalent *deterministic Turing machine*. Recall that we used a *DTM* with three tapes to do this.

2. [2 points] What did we use tape 1 for? Did we alter this tape? If so, how and why?

3. [2 points] What did we use tape 2 for? Did we alter this tape? If so, how and why?

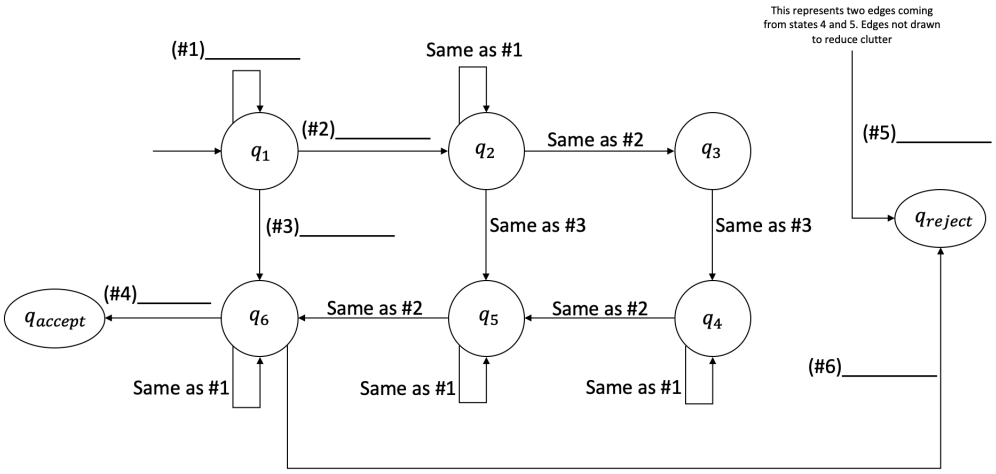
4. [2 points] What did we use tape 3 for? Did we alter this tape? If so, how and why?

The following question asks you to consider a *Turing Machine* that recognizes the following language:

$L = \{X\#Y | \Sigma = \{a, b\}, |X| = |Y| = 2,$ and the number of a's in X and Y is equal$\}$

In other words, given two unique length two strings separated by a pound symbol, accept iff each length two string contains the same number of a's. For example, $ab\#ba \in L$, $bb\#bb \in L$, $aa\#aa \in L$, but $aa\#ab \notin L$.

5. [6 points] Given the *Turing Machine* below, fill in the blanks to complete the implementation. All transitions should be of the form input arrow, move head (e.g., $a \rightarrow L$). You will never have to write to the tape. Many of the transitions are the same. The machine labels transitions that are equivalent so that you only need to fill out one blank. Note that some blanks may still be the same even if they are separately labeled below as questions. Also note that this machine assumes the input is always formatted correctly for simplicity. *Hint: States 1 through 3 are counting the number of a's found in the first two characters, then we move to states 4-6 which count the number of a's in the second string.*



NOTHING BELOW THIS POINT WILL BE GRADED